The PSI3 User's Manual

Jeffrey C. Stephens, George Vacek,
C. David Sherrill, Justin T. Fermann,
Timothy J. Van Huis, and T. Daniel Crawford
Revised for PSI3 by Edward F. Valeev and C. David Sherrill

Center for Computational Quantum Chemistry
University of Georgia
Athens, Georgia 30602

Center for Computational Molecular Science and Technology
Georgia Institute of Technology
Atlanta, GA 30332-0400

Department of Chemistry
Virginia Tech
Blacksburg, VA 24061-0212

Version of: January 20, 2003

Contents

1 Introduction **4**

2 PSI3 Preliminaries **4**

2.1 Capabilities 4

2.2 Before You Obtain PSI3: Prerequisites 4

2.3 Before You Run PSI3 5

3 PSI3 Tutorial **6**

3.1 Running PSI3 6

3.2 Format of input.dat 6

3.3 Setting up a Calculation 9

3.4 Carrying Out a Calculation 12

4 Running PSI3 **13**

4.1 Geometry Specification 13

4.2 Molecular Symmetry 15

4.3 Basis Sets 16

4.3.1 Default Basis Sets 16

4.3.2 Custom Basis Sets 18

4.4 Electronic Structure Specification 19

4.5 Single-Point Energy Computation 20

4.6 Geometry Optimization 20

4.6.1 Internal Coordinates and Structure of intco Vector 21

4.6.2 Force Constant Matrix and Structure of fconst.dat 23

4.7 Frequency Analysis 23

4.8 Evaluation of one-electron properties 24

4.9 Plotting one-electron properties 24

4.10 Utilities 26

4.10.1 geom 26

5 PSI3 modules **26**

5.1 Overview of modules of PSI3 26

5.1.1 input 26

5.1.2 cints 26
5.1.3 cscf 26
5.1.4 transqt 27
5.1.5 ccsort 27
5.1.6 ccenergy 27
5.1.7 cctriples 27
5.1.8 detci 27
5.1.9 detcas 27
5.1.10 detcasman 27
5.1.11 clag 27

A `PSI3` Reference **28**

B Sample `input.dat` file **28**

C Sample `basis.dat` file **30**

1 Introduction

This manual is intended to provide users with up-to-date information on running the PSI3suite of *ab initio* quantum chemical programs. In section 2 we provide an overview of some of the features of PSI3 along with the prerequisite steps for running calculations. Section 3 provides a brief tutorial to help new users get started. Section 4 offers further details into the structure of PSI3 calculations, including descriptions of the most important options and modules. Later sections deal with more advanced aspects of PSI3 calculations, including cutomization of basis sets, scratch disk space, and troubleshooting. The appendix includes a description of the input keywords and command-line options for each module, as well as numerous examples of PSI3 input and basis set files.

Instructions for obtaining, compiling, and installing PSI3 on UNIX systems are given in the separate Installation Manual. The latest PSI3 documentation can be found at `http://vergil.chemistry.gatech.edu/psi/`

2 PSI3 Preliminaries

2.1 Capabilities

PSI3 can perform *ab initio* computations employing basis sets of up to 32768 contracted Gaussian-type functions of virtually arbitrary orbital quantum number. PSI3 can recognize and exploit the largest Abelian subgroup of the point group describing the full symmetry of the molecule. Table 1 displays the range of theoretical methods available in PSI3. Geometry optimization (currently restricted to true minima on the potential energy surface) and vibrational frequency computations can be performed with the methods for which analytic gradients and Hessian, respectively, are available. Methods for which analytic gradients are not yet implemented may still be used for geometry optimization if the gradients are computed numerically from energies; this is automated by the optking program. Numerical computation of Hessians is being developed. PSI3 can also compute an extensive list of one-electron properties. Finally, it should be mentioned that whenever PSI3 is used, it should be cited fully (see appendix).

2.2 Before You Obtain PSI3: Prerequisites

The majority of PSI3 was developed on IBM RS/6000//AIX and Intel i386/Linux workstations. The complete list of tested processor/OS combinations to which PSI3 has been ported is shown in Table 2. If you don't find your system in the Table, there's a good chance that you will be able to install PSI3 on your system if you have all of the following:

Table 1: Summary of theoretical methods available in PSI3.

Method	Energy	Gradient	Hessian
RHF SCF	Y	Y	Y
ROHF SCF	Y	Y	N
UHF SCF	Y	N	N
CIS/RPA/TDHF	Y	N	N
TCSCF	Y	Y	N
CASSCF	Y	Y	N
RAS-CI	Y	N	N
RHF-MP2	Y	N	N
RHF-MP2-R12	Y	N	N
RHF-CCSD	Y	Y	N
ROHF-CCSD	Y	Y	N
VOO-CCD	Y	Y	N
RHF-CCSD(T)	Y	N	N
UHF-CCSD(T)	Y	N	N
RHF-EOM-CCSD	Y	N	N
ROHF-EOM-CCSD	Y	N	N

Table 2: Platforms on which PSI3 has been installed successfully.

Hardware	Operating System(s)	Special Notes
Compaq Alpha	Compaq TrueUNIX64	64-bit mode
IBM RS/6000	AIX 3.2.5, AIX 4.1-4.3.0	IBM C, C++, and Fortran compilers
IBM PowerPC	AIX 4.3.2	IBM C, C++, and Fortran compilers; 64-bit mode
Intel ix86	Linux 2.2	
SGI Origin 2000	IRIX64 6.5	64-bit

2.3 Before You Run PSI3

Every user needs to configure her or his shell environment prior to running PSI3:

1. add the location of the binaries to $PATH
2. add the location of the manpages to $MANPATH

In C-shell it may be achieved like this:

```
set psipath = /put/psi/installation/directory/here
setenv PATH $psipath‘$psipath/bin/host.sh‘/bin:$PATH
setenv MANPATH $psipath/doc/man:$MANPATH
```

Note that in previous versions of PSI, the installation directory was frequently in a subdirectory of the source distribution directory. This is no longer the case. Now, `psipath` is commonly something like `/usr/local/psi3` while the source directory will be elsewhere.

3 PSI3 Tutorial

3.1 Running PSI3

The PSI3 suite of codes is built around a modular design which allows it great power and flexibility. Any module can be run independently (provided suitable datafiles, of course). There also exists a master program, appropriately called `psi`, which will parse an input file, recognize the overall calculation desired, and run all the necessary modules in the correct order. To begin with

```
c4-20:> man psi
```

will get the online manual page for the `psi` master program. Other man pages for the `psi` code can be retrieved likewise.

The only required file is to run PSI3 is `input.dat`. Unfortunately, no good and simple description of this file exists. It is rather simple though, and essentially free format. Examples can be found in almost every man page. It should be fairly easy to understand all the examples using common sense.

3.2 Format of input.dat

Some keys to remember are the following:

- Input parsing is case insensitive; once parsed, everything is treated as upper case.
- Anything following a percent mark, i.e. %, is commented out up until the next carriage return.
- White space of more than a single space is ignored.
- Anything between double quotes, e.g. "/usr/c4/sw/psi", is considered one token; there is no change of case, and special characters and white space are maintained as part of the token but otherwise ignored.
- The old (Fortran) input parser does not like the TAB character. Please avoid using Tabs until this goes away. There does not appear to be a problem with the C library (libipv1).

Input data types are the following:

- String : a character sequence
- Integer : a `sizeof(integer)` byte integer datum
- Real : a `sizeof(double)` byte real datum
- Boolean : `yes`, `true`, `1` – these three values are equivalent; `no`, `false`, `0` – these three values are equivalent

Input parameters come in only a few flavors:

- `Keywords = value`, e.g. `convergence = 12`
- `Vectors = (value1 value2 ... valuen)`, e.g. `docc = (2 0 1 1)`
- `Arrays = ((i1 i2 ... in) (j1 j2 ... jn) ... (n1 n2 ... nn))` Note the proper number of opening and closing parentheses. Arrays can conceptually be any depth/dimension but in practice never go much beyond level 2. The elements of vectors and arrays need not be of the same data type.

Segmented Keywords are joined by a colon, i.e. ":".

```
scf:convergence = 12
gugaci:convergence = 8
```

Segmented keywords with common initial segments can be joined in vectors.

```
scf:(convergence = 12
     docc = (2 0 1 1))
```

Note the equivalence of the following three examples with the preceding one.

```
scf:(convergence = 12 docc = (2 0 1 1)
scf:( convergence=12  docc=(2 0 1 1))
scf: (
   convergence = 12
   docc = (2 0 1 1)
)
```

A "default" token can be used as a wild card, to fill any initial token field. However, explicit tokens will always override default tokens if both are present. For instance, a typical keyword search priority would be: First

```
module_id:parameter
```

and then

```
default:parameter
```

Thus the two following examples will achieve the same effect when the module `transqt` is run, but not when `cscf` is run.

```
default: (
  convergence = 8
  docc = (2 0 1 1)
)
scf: (
  convergence = 12
)
```

```
default: (
  convergence = 8
  docc = (2 0 1 1)
)
scf: (
  convergence = 10
)
```

Ok, enough of all that, you've got the basics down, the rest can be learned by doing. The smallest `input.dat` file you would probably ever want to use, and this only as a learing experience, is:

```
% file input.dat
psi: (
  check = true
  )
% end file input.dat
```

If we run psi now, what happens?

```
c4-20:> psi
                    The Psi Execution Driver
 ERROR: a problem arose while reading the required string valued keyword 'WFN'
c4-20:>
```

Obviously we need to add a `WFN`. So

```
% file input.dat
psi: (
  check = true
  wfn = scf          % for instance
  )
```

```
% end file input.dat

c4-20:> psi
                    The Psi Execution Driver
 WFN       = SCF
 DERTYPE   = NONE
 REFERENCE = RHF
 CHECK     = YES

 'CHECK' is YES, so nothing will be executed.
 The following programs would otherwise be executed:
 cints
 cscf
c4-20:>
```

As you can see, some defaults are chosen. We are now doing a RHF SCF Energy Point. First the module (or program) `cints` will run to calculate the integrals, then `cscf` will run to calculate the RHF SCF energy.

Should we run this, just for the learning experience? Sure, but first we need to do a couple things. Like find out what files and input parameters are needed by `cints` and `cscf`.

```
c4-20:> man cints
c4-20:> man cscf
```

`cints` needs `input.dat` and `file30` and will generate `file33`, `file35`, `file36`, and `file37`. It requires only 2 input parameters and the defaults look ok. `cscf` needs `input.dat`, `file30`, `file33`, `file35`, `file36`, and `file37` and a few input parameters which we will come back to.

So what's this `file30`? It is the basic binary working file for the `PSI3` quantum chemistry package. It contains all the juicy details of your job, such as geometry, basis set, occupation scheme, etc. Since no one likes to write binary files, we will create `file30` with the program `input`. For more information on this program

```
c4-20:> man input
```

3.3 Setting up a Calculation

I've decided that we'll start with STO water for our example, and after reading the man pages described above I come up with the following:

```
% file input.dat
default: (
```

```
  label = "water STO HF energy point"
  memory = (8.0 Mbytes)
  wfn = scf
  reference = rhf

  files: (
    default: (
      name = "h2osto"
      nvolume = 4
      volume1 = "/tmp1/psiuser/"
      volume2 = "/tmp2/psiuser/"
      volume3 = "/tmp2/psiuser/"
      volume4 = "/tmp2/psiuser/"
      )
    file30: (
      nvolume = 1
      volume1 = "./"
      )
    )
  )

psi: (
  check = true
  )

input: (
%    note that all atoms are specified now,
%    not only the symmetry unique portion
  basis = sto-3g
  units = angstrom
  zmat = (
    (o)
    (h 1 0.9600)
    (h 1 0.9600 2 104.5)
    )
  )
% end file input.dat
```

You may wonder about the memory flag I've tucked in the default. As far as I know, it's undocumented, but can be set for some PSI modules. Acceptable units are REAL, INTEGER, BYTES, KBYTES, or MBYTES. Generally it is a good practice to keep `default:memory` small, and increase `program_id:memory` as necessary.

The `default:files` section tells `PSI3` where to store temporary (scratch) files. Here, a user is telling `PSI3` to write the scratch files to four different directories (where in this case "psiuser" is used as the user's username). The file-handling capability of `PSI3` is very

flexible — it is possible, for example, to tell each module to place files in different locations (although this would rarely ever be useful). By placing a `files` section in the `default` area, `PSI3` will use this information by default for all of its modules unless overridden by a `files` section inside a particular module (like `cscf`, for example). Specifying four volumes, such as in this example, tells `PSI3` to stripe (split the files) over four different directories. This can be useful if there are four hard disks, all used for scratch data. However, modern operating systems allow different physical hard drives to be striped automatically by the operating system into a single directory, so having `nvolume` more than 1 is pretty unusual nowadays.

One downside to listing these specific scratch directories is that, if you send your `PSI3` input file to a co-worker, he or she will have to modify the file to use their scratch directories instead of yours (assuming you don't all write to the same directory, like `/tmp`). This problem is solved by the `.psirc` file. If this file exists in the user's home directory, then it will override the `file` information in `input.dat`. For example, with a `.psirc` file containing the following information,

```
default: (
    files: (
      default: (
        nvolume = 4
        volume1 = "/tmp1/psiuser/"
        volume2 = "/tmp2/psiuser/"
        volume3 = "/tmp2/psiuser/"
        volume4 = "/tmp2/psiuser/"
      )
      file30: (
        nvolume = 1
        volume1 = "./"
      )
    )
)
```

then the relevant section in `input.dat` can be vastly simplified to just

```
  files: (
    default: ( name = "h2o" )
    )
```

Once you have a `input.dat` file prepared, the first thing to do is to create the checkpoint file, the previously mentioned `file30`. You can do this by running the program

```
  c4-20:> input
```

You should now have two new files `output.dat` and `h2osto.30`. The latter file is a binary file, it cannot be directly examined. The first file should contain text giving the program output. Take a look at this file and see what kind of information it gives.

Going on,

```
c4-20:> psi
```

reminds us that we will be running `cints` and `cscf`. We have everything we need for the first, but how about the second.

```
c4-20:> man cscf
```

Well, there are alot of possibilities, but the defaults are generally sound, so we really only need to add in our occupation scheme for the doubly-occupied orbitals. Or do we? The fact is `cscf` can guess occupations for you using core Hamiltonian orbital eigenvalues, but, as with any computer program, you have to be cautious. Let us be adventurous here and let the program guess. In general, always make sure you are computing the state you want to and not the state that the program chose.

Notice at this point that I am leaving out the majority of the file `input.dat`. It is getting a little big to repetitiously include it within this document.

3.4 Carrying Out a Calculation

Assuming you have done everything detailed in the previous section, we can run an actual job and get a number out.

```
c4-20:> psi
```

Ahhhh. We forgot to change the `psi:check=true` flag, so nothing was really run. Change it to `psi:check=false`, or simply comment it out (since the default value is false) by inserting a " `%`" at the start of that line, and run psi again. Take a look at the file `output.dat` and become familiar with it.

We now have the basic `input.dat` file. Only minor modifications will allow it to be used to run a great variety of jobs. First of all let's move on to another basis set. The only thing that needs to be changed is the parameter `input:basis=sto-3g` to, for instance, `input:basis=dz`. But for consistency, why not change `default:label` and `default:files:default:name`?

```
c4-20:> input
c4-20:> psi
```

Whenever you change the basis parameter, you must run input so that the change in basis is included into `file30`. Very important. Changing `default:files:default:name` every time you change the basis parameter is a good habit, as it will insure that you run `input`. Do you see why?

An aside about `cscf` and guess wavefunctions: when you run `cscf`, it (by default, but can be overridden) automatically checks to see if there is an old wavefunction in `file30`

which can be used as a guess wavefunction, and if so, it uses it. Each time `cscf` completes, (converged or not) it writes the final wavefunction to `file30`. Each time `input` is run, it overwrites `file30` and any wavefunction that might have been in there is lost. Nothing tricky about any of this, but just something to keep in mind. If you want to re-use MO's from a previous calculation as a guess, you can run `input` with the `--chkptmos` option. This will work even if the previous calculation used a different basis set, but the geometry and point group symmetry should be the same. Another useful flag is `--noreorient`, which prevents the molecule from reorienting after shifting the origin to the center-of-mass.

Well, energy points are well and good, but without optimizations, we are not going to get very far. So, we need `psi:dertype=first`, `psi:opt=true`, and `psi:nopt=7` to run up to 7 cycles of geometry optimization (Or we could set `default:dertype=first` and `default:opt=true`, or even mix and match). Set `psi:check=true` and run `psi` to see what modules will be run. Then check the man pages for each module and see what you need for it to run. For instance, in this case we see that `cints --deriv1` will be run. `man cints` shows that us that everything is set OK. `optking` is the last program to be run in each optimization cycle. `PSI2` users will appreciate the ease with which it can optimize molecules as it can generate internal coordinates for you as well as guess a force constant matrix. Hence defaults are sufficient to run this program.

OK! Go ahead and run `psi`. In the 5th cycle `optking` should return a non-zero value and `psi` will stop the procedure. It will indicate that the optimization is over. Let's up the ante and go for an optimization with a DZP basis set.

You know the procedure for improving the basis set. Just switch the label, the files name and the basis parameter. Oh, before I forgot, you have to remove file named `opt.aux`. Then run `input` and `psi`. The job runs fine. Go ahead and look at `output.dat`. You may also wish to look at `file11.dat`. `fconst.dat` contains now an improved force constant matrix. `optking` will automatically use this improved FC matrix if it is present.

OK, the calculation did converge in 4 steps. What if it didn't? Just increasing the number of optimization steps might just do that but might not. In general, 10 optimization steps should be enough for anyone. If it doesn't converge by then, your initial guess geometry or initial FC matrix or both were bad. Rethink your situation. Oops, this is supposed to be a howto-psi, not a QC-theology.

4 Running `PSI3`

4.1 Geometry Specification

Full molecular geometry has to be specified in form of Cartesian coordinates or a Z-matrix. Cartesian coordinates of atoms are specified via a keyword `geometry` which has to be a member of either `default` or `input` sections:

```
geometry = (
  (atomname1 x1 y1 z1)
```

```
  (atomname2 x2 y2 z2)
  (atomname3 x3 y3 z3)
      ..........
  (atomnameN xN yN zN)
)
```

where `atomname`*i* can take the following values:

- element symbol (H, He, Li, Be, B, etc.);
- full element name (hydrogen, helium, lithium, etc.);
- *ghost* atom symbol (G) or name (ghost). Ghost atom is an atom of formal charge 0.0, it can be useful to specify the location of the off-nucleus basis functions;
- *dummy* atom symbol (X). Dummy atoms can be useful only to specify Z-matrices of proper symmetry (not used in `PSI3`; see below) or which contain linear fragments.

Hence the following two examples are equivalent to one another:

```
geometry = (
  (H 0.0 0.0 0.0)
  (f 1.0 0.0 0.0)
  (Li 3.0 0.0 0.0)
  (BE 6.0 0.0 0.0)
)

geometry = (
  (hydrogen  0.0 0.0 0.0)
  (FLUORINE  1.0 0.0 0.0)
  (Lithium   3.0 0.0 0.0)
  (berillium 6.0 0.0 0.0)
)
```

The keyword `units` specifies the units for the coordinates:

- `units = bohr` – atomic units (Bohr), default;
- `units = angstrom` – angstroms ($\mathring{A}$);

The `zmat` keyword can be used to specify a Z-matrix for the molecule. It also has to be put in either `default` or `input` sections. The format of this vector is as follows:

```
zmat = (
  (atomname1)
  (atomname2 ref21 bond_dist2)
```

```
  (atomname3 ref31 bond_dist3 ref32 bond_angle3)
  (atomname4 ref41 bond_dist4 ref42 bond_angle4 ref43 tors_angle4)
  (atomname5 ref51 bond_dist5 ref52 bond_angle5 ref53 tors_angle5)
                   ........................
  (atomnameN refN1 bond_distN refN2 bond_angleN refN3 tors_angleN)
)
```

where

- `bond_dist`i is the distance (in units specified by keyword `units`) from nucleus number i to nucleus number `ref`i`1`. The units
- `bond_angle`i is the angle formed by nuclei i, `ref`i`1`, and `ref`i`2`;
- `tors_angle`i is the torsion angle formed by nuclei i, `ref`i`1`, `ref`i`2`, and `ref`i`3`;

Some care has to be taken when constructing a Z-matrix for a molecule which contains linear fragments. For example, let's construct a Z-matrix for a linear conformation of HNCO. The first three atoms (HNC) can be specified as is, but the fourth atom (O) poses a problem – the torsional angle cannot be defined with respect to the linear HNC fragment. The solution is to add 2 dummy atoms to the definition:

```
zmat = (
  (h)
  (n 1 1.012)
  (x 2 1.000 1  90.0)
  (c 2 1.234 3  90.0 1 180.0)
  (x 4 1.000 2  90.0 3 180.0)
  (o 4 1.114 5  90.0 2 180.0)
)
```

Of course, a choice of the method for geometry specification is solely a matter of convenience.

4.2 Molecular Symmetry

`PSI3` can determine automatically the largest Abelian point group for a valid framework of centers (the framework also includes ghost atoms, but it does not include dummy atoms). It will then use the symmetry porperties of the system in computing energy, forces, and other properties. However, in certain instances it is desirable to use a lower than the full symmetry. The keyword `subgroup` is used to specify a subgroup of the full molecular point group. The allowed values are `c2v`, `c2h`, `d2`, `c2`, `cs`, `ci`, and `c1`. For certain combinations of a group and its subgroup there is no unique way to determine which subgroup is implied. For example, D_{2h} has 3 non-equivalent C_{2v} subgroups, e.g. $C_{2v}(X)$ consists of symmetry operations $\hat{E}$, $\hat{C}_2(x)$, $\hat{\sigma}_{xy}$, and $\hat{\sigma_{xz}}$. To specify subgroups precisely one has to use the `unique_axis` keyword.

E.g. the following input will specify the $C_{2v}(X)$ subgroup of D_{2h} to be the computational point group:

```
input: (
  geometry = (
    ........
  )
  units = angstrom
  subgroup = c2v
  unique_axis = x
)
```

4.3 Basis Sets

PSI3 uses basis sets comprised of Cartesian (`puream = false`, default) or Spherical Harmonic (`puream = true`) Gaussian basis functions. An atomic basis set is normally identified by a string. Currently, there exist three ways to specify which basis sets to use for which atoms:

- `basis = string` – all atoms use basis set type. If the basis string contains any "special" characters (e.g., parentheses, asterisks) then the string must be enclosed in quotation marks, e.g., "6-311++G(d,p)".
- `basis = (string1 string2 string3 ... stringN)` – `string` *i* specifies the basis set for atom *i*. Thus, the number of strings in the `basis` vector has to be the same as the number of atoms (including ghost atoms but excluding dummy atoms). Another restriction is that symmetry equivalent atoms should have same basis sets, otherwise `input` will use the string provided for the so-called unique atom out of the set of symmetry equivalent ones.
-
```
basis = (
  (element1 string1)
  (element2 string2)
     ...........
  (elementN stringN)
)
```

  `string` *i* specifies the basis set for chemical element `element` *i*.

4.3.1 Default Basis Sets

PSI3 default basis sets are located in `pbasis.dat` which is located in `$psipath/share`. Table ?? lists basis sets defined in `pbasis.dat`.

Table 3: Basis sets available in PSI 3.2

Basis Set	Atoms	Aliases
Huzinaga-Dunning		
(4S/2S)	H	
(9S5P/4S2P)	B-F	
(11S7P/6S4P)	Al-Cl	
DZ	H, B-F, Al-Cl	
DZP	H, B-F, Al-Cl	
DZ-DIF	H, B-F, Al-Cl	
DZP-DIF	H, B-F, Al-Cl	
Wachters		
WACHTERS	K, Sc-Cu	
WACHTERS-F	Sc-Cu	
Pople-type		
STO-3G	H-Ar	
3-21G	H-Ar	
6-31G	H-Ar, K, Ca, Cu	
6-31G*	H-Ar, K, Ca, Cu	6-31G(d)
6-31+G*	H-Ar	6-31+G(d)
6-31G**	H-Ar, K, Ca, Cu	6-31G(d,p)
6-311G	H-Ar	
6-311G*	H-Ar	6-311G(d)
6-311+G*	H-Ne	6-311+G(d)
6-311G**	H-Ar	6-311G(d,p)
6-311G(2df,2pd)	H-Ne	
6-311++G**	H, B-Ar	6-311++G(d,p)
6-311G(2d,2p)	H-Ar	
6-311++G(2d,2p)	H-Ar	
6-311++G(3df,3pd)	H-Ar	
Triple-Zeta		
TZ2P	H, B-F, Al-Cl	
TZ2PD	H	
TZ2PF	H, B-F, Al-Cl	
TZ-DIF	H, B-F, Al-Cl	
TZ2P-DIF	H, B-F, Al-Cl	
TZ2PD-DIF	H	
TZ2PF-DIF	H, B-F, Al-Cl	
Correlation Consistent (N = D,T,Q,5,6)		
CC-PVNZ	H-Ar	cc-pVNZ
CC-PV(N+D)Z	Al-Ar	cc-pV(N+d)Z
CC-PCVNZ	B-Ne	cc-pCVNZ
AUG-CC-PVNZ	H-He, B-Ne, Al-Ar	aug-cc-pVNZ
AUG-CC-PV(N+D)Z	Al-Ar	aug-cc-pCV(N+d)Z
AUG-CC-PCVNZ (N<6)	B-F	aug-cc-pCVNZ
D-AUG-CC-PVNZ	H	
PV7Z	H, C, N, O, F, S	pV7Z

4.3.2 Custom Basis Sets

To make a custom basis set, enter the information in either of the following four files:

- `pbasis.dat` – only if you think it should be added to `PSI3`. You also might want to check in your additions and changes so that everyone could benefit from them. Refer to the `PSI3` Programmer's Manual for information on how to access `PSI3` repository.
- An arbitrary text file. To specify the file's location use keyword `basisfile`:

```
input: (
  % The meaning of this is pretty obvious
  basisfile = "/home/users/tool/chem/h2o/mybasis.in"
  % If the location ends with '/', "basis.dat" is automatically appended
  % Hence this specifies /home/users/tool/chem/basis.dat !
  basisfile = "/home/users/tool/chem/"
)
```

  Use this option if you want to use the basis set file in a project which involves running more than one computation.

- File named `basis.dat`, which resides in the working directory along with `input.dat`. Same use as the previous entry.
- `input.dat`

The order in which `input` program searches for basis sets is the order in which files appear in our checklist.

A contracted Cartesian Gaussian-Type Orbital

$$\phi_{\mathrm{CGTO}} \;=\; x^l y^m z^n \sum_i C_i \exp(-\alpha_i [x^2 + y^2 + z^2]) \tag{1}$$

where

$$L \;=\; l + m + n \tag{2}$$

is written as

```
basis: (
  ATOM_NAME: "BASIS_SET_LABEL" = (
    (L (C1  alpha1)
       (C2  alpha2)
       (C3  alpha3)
       ...
       (Cn  alpha4))
    )
  )
```

To scale a basis set, a scale factor may be added as the last item in the specification of each contracted Gaussian function. For example, to scale the S functions in a 6-31G** basis for hydrogen, one would use the following

```
hydrogen:"6-31G**" =
    ( (S (    18.73113696     0.03349460)
         (     2.82539437     0.23472695)
         (     0.64012169     0.81375733) 1.2 )
      (S (     0.16127776     1.00000000) 1.2 )
      (P (     1.10000000     1.00000000))
     )
```

In this example, both contracted S functions have their exponents scaled by a factor of $(1.2)^2$ = 1.44. The output file should show the exponents after scaling.

4.4 Electronic Structure Specification

The reference electronic configuration of a molecule is specified via a combination of keywords `reference` and `multiplicity` and occupation vectors `docc` and `socc`. However, the latter may not be necessary as `cscf` may guess occupations (`docc` and `socc` arrays) for you had `charge`, `multiplicity`, and `reference` have been specified. It is the easiest way to specify electronic configuration for your system, but remember that guessing algorithms `cscf` uses are far from perfect. Hence you should check guessed occupations every time you let `cscf` guess them for you.

To determine the electronic occupations in `PSI3` manually, first construct symmetry orbitals using group theory and fill them according to regular valence bond arguments. To define your occupations in `PSI3`, use the `docc` and `socc` arrays. But only `docc` and `socc` may not be enough to specify precisely the spin couplings in your system. That's where `reference` and `multiplicity` keywords come in. `multiplicity` is equal 2S+1, where S is the spin quantum number of the system. `reference` can equal

- rhf (default) - spin-restricted reference for closed shell molecules. `multiplicity` may only equal to 1 in this case.
- rohf - spin-restricted reference for open shell molecules. If multiplicity=1 and socc has two singly occupied orbitals in different symmetry blocks - it's equivalent to the old opentype=singlet statement. Otherwise it's assumed to be a high-spin open-shell case (equivalent to the old opentype=highspin statement).
- uhf - spin-unrestricted reference for closed shell or high-spin (parallel spins) open shell system.
- twocon for two determinantal wavefunctions. The largest component should be specified by the docc and socc arrays. Multiplicity has to be set to 1.

For $^1A_1$ methylene, the occupation is (1a1)2(2a1)2(1b2)1(3a1)2 so the docc is:

```
reference = rhf or uhf
multiplicity = 1
docc = (3 0 0 1)
```

For the $^3B_1$ state of methylene, the electronic configuration is (1a1)2(2a1)2(1b2)2(3a1)1(1b1)1 so the docc and socc arrays are:

```
reference = rohf or uhf
multiplicity = 3
docc = (2 0 0 1)
socc = (1 0 1 0)
```

For the $^1B_1$ state of methylene however, the docc and socc arrays are also:

```
reference = rohf
multiplicity = 1
docc = (2 0 0 1)
socc = (1 0 1 0)
```

Since most of the basis sets are highly contracted in the core regions, core electrons are routinely frozen and corresponding virtual orbitals are deleted. This is accomplished via the `frozen_docc` and `frozen_uocc` arrays. Simply specify the symmetry of the frozen orbital and `PSI3` will do the rest.

To freeze the lowest a_1 orbital and delete the corresponding highest b_2 orbital, they would look like this:

```
frozen_docc = (1 0 0 0)
frozen_uocc = (0 0 0 1)
```

4.5 Single-Point Energy Computation

Along with the wavefunction type, the nuclear framework, basis set, and electronic configuration are sufficient for a single-point evaluation of the electronic energy. The electronic wavefunction is specified via the `wfn` keyword. The range of allowed wavefunctions is listed in Table 1.

4.6 Geometry Optimization

`optking` is the program responsible for orchestrating the process of geometry optimization. It can do a number of tasks automatically, such as generating internal coordinates, produce empirical force constant matrix, if necessary, update it, and check if geometry optimization is over. Some or all of the following files are necessary to perform a geometry optimization with `optking`:

- `file11.dat` - contains the cartesian geometry and the nuclear forces, produced by `cints --deriv1`;
- `fconst.dat`- contains force constants; if absent - empirical force constants will be generated by `optking`;
- `intco.dat`- contains internal coordinates in a format readable by a human; if absent - internal coordinates are generated automatically by `optking`.

The procedure for setting up such a calculation is as follows:

- define internal coordinates if desired (or, do nothing, and `optking` will do it for you automatically!)
- obtain a set of force constants in an fconst.dat file (or, again, `optking` can do this automatically for you)
- If analytic gradients are available for your chosen method, set `dertype=first`. If not, set `dertype=none` and also set `numerical_dertype=first` in the `default` section.
- Run the optimization by setting the `opt` flag set to true and `nopt` to the number of geometry optimization steps (say, around 5 to 10). If analytic gradients are not available, then `nopt` instead gives the number of energy points to compute. This should be the desired number of geometry optimization steps, multiplied by (2*num_symm_coord + 1), where num_symm_coord is the number of totally-symmetric internal coordinates.

The precision with which geometry is optimized depends on the residual forces on the nuclei. By default `optking` will terminate the job if the residual cartesian gradients in `file11.dat` are less than 1E-5 in atomic units. It is probably enough for most tasks. Going below this will most likely waste CPU time unless you are doing benchmarks.

An important aspect of a geometry optimization is the accuracy of the first derivatives of energy that `PSI3` computes. Depending on how poorly your wavefunction has been convereged, the gradients themselves may not be sufficiently accurate for the requested convergence criterion. After computing first derivatives of the energy, `cints` runs a simple check of the quality of the energy derivative. It's a good idea to look at `cints`' output to make sure that the gradients are OK.

Let us take a look at each step involved in optimizing molecular geometry.

4.6.1 Internal Coordinates and Structure of `intco` Vector

This section is largely obsolete now with the addition of the `optking` program which can generate internal coordinates automatically. At present `optking` cannot handle molecules larger than a few atoms but it should change in the immediate future. Hence you may still specify internal coordinates manually as described here, but this ability may become obsolete someday.

PSI3 currently carries out all optimizations in internal coordinates. The internals are specified in either `input.dat` or `intco.dat`. First, the primitive internals are defined. These are individual stretches, bends, torsion, out-of-plane deformations, and two different linear bends denoted lin1 and lin2. All of these are defined in Wilson, Decius, and Cross. An example for methane is below:

```
intco: (
   stre = (
     (1 1 2)
     (2 1 3)
     (3 1 4)
     (4 1 5)
   )
   bend = (
     (5 2 1 5)
     (6 3 1 5)
     (7 4 1 5)
     (8 2 1 4)
     (9 3 1 4)
     (10 2 1 3)
   )
```

After the primitives are defined, they are constructed into symmetrized internals with the totally symmetric placed in the SYMM vector and the rest placed in the ASYMM vector. For optimizations, only the SYMM internals need to be defined. Likewise, if during an optimization a molecule breaks symmetry, the internals have been improperly defined. Again, methane is done below:

```
    symm = (
     ("(1) stretch"(1 2 3 4))
   )
    asymm = (
     ("(2) E bend"(10 7))
     ("(3) T2 stretch"(1  2  -3  -4 ))
     ("(4) T2 bend"(10 -7))
     ("(5) E torsion"(8 -5  -9  6))
     ("(6) T2 bend" (8 -6))
     ("(7) T2 bend"( 5 -9))
     ("(8) T2 stretch"(1  3  -2  -4))
     ("(9) T2 stretch"(1  4 -2  -3))
   )
)
```

The SYMM and ASYMM vectors have two or three components: the first is a label enclosed by quotation marks and the second is the list of primitive internals comprising this vector.

Some internals have been multiplied by -1 to reflect the appropriate symmetries. If the internals need to be weighted by some prefactor, then a third vector may be used:

```
symm = (
 ("generic coord" (1 -2 -3) (2.0 1.0 1.0))
)
```

For more information in defining symmetric internals, refer to Cotton's text.

4.6.2 Force Constant Matrix and Structure of `fconst.dat`

The quality of the force constants, or Hessian, is critical for optimizing weakly bound structures. In order to start an optimization, one needs the `fconst.dat` file. For those of you that can speak Fortran 77, this file is written in 8F10.7 format. It is the lower triangle of the force constant matrix in internal coordinates. The order of the forces is identical to the order of the SYMM and ASYMM vectors. For the methane-water dimer, an excerpt from a real `fconst.dat` is shown below:

```
5.654908
 .217027  5.616085
-.006096  -.001145  6.078154
-.055291   .026921   .023732   .317485
 .004063  -.155489  -.003880  -.170201   .873999
-.146900  -.285108   .001153  -.023346   .196008   .605239
 .001037  -.001959   .002945  -.024597   .014432   .005302   .388622
```

Ideally, your diagonal elements should be the much larger than the non-diagonal elements. If you need an `fconst.dat` file, you have four options:

1. Create a diagonal matrix of 1's
2. Create a diagonal matrix with 5 for stretching coordinates, 2 for bending coordinates, and 1 for all other coordinates
3. Let `optking` generate an empirical Hessian for you
4. Run a second derivative to obtain a Cartesian Hessian and transform that to internals(fconst.dat) with intder.

Clearly, the list starts at the most approximate and gets more accurate.

4.7 Frequency Analysis

Currently, analytic second derivatives are avaliable only for closed-shell Hartree-Fock (the preliminary implementation is not yet very efficient). For methods with analytic gradients,

one must carry out finite difference calculations to obtain second derivatives. If one has the misfortune of needing frequencies for methods where only energies exist(e.g. FCI or MP2-R12), finite differences with gradients approximated by energy points must be used. At present, obtaining the Hessian numerically from energies or gradients is not automated by the program. There are two types of frequency analysis programs available within `PSI3`, `normco` and `intder95`. For more details regarding these programs, see the manual pages for each program respectively.

4.8 Evaluation of one-electron properties

For now, take a look at the available documentation for `oeprop`.

4.9 Plotting one-electron properties

Program `oeprop` can evaluate certain one-electron properties on a grid of points and then print them out in format suitable for image rendering with external programs. Currently, electron and spin density, electron and spin density gradient, Laplacian of electron and spin density, and electrostatic potential can be evaluated on an arbitrary rectangular two-dimensional grid and output in a format suitable for feeding to a interactive visualization program `PlotMTV` (version 1.3 and higher). `PlotMTV` is a freeware code developed by Kenny Toh. It can be downloaded off many web sites in source or binary form.

In addition, values of molecular orbitals can be evaluated on an arbitrary rectangular three-dimensional grid and output for furher rendering of high-quality images with a program `MegaPov` (version 0.5). `MegaPov` is an unofficial patch for a ray-tracing code `POV-Ray`. Information on `MegaPov` can be found at http://nathan.kopp.com/patched.htm.

Let's look at how to set up input for spin density evaluation on a two-dimensional grid. An input secion of `oeprop` might look like this:

```
oeprop:(
  grid = 2
  spin_prop = true
  grid_origin = (0.0 -5.0 -5.0)
  grid_unit_x = (0.0 1.0 0.0)
  grid_unit_y = (0.0 0.0 1.0)
  grid_xy0 = (0.0 0.0)
  grid_xy1 = (10.0 10.0)
  nix = 20
  niy = 20
)
```

`grid` specifies the type of a property and the type of a grid `oeprop` needs to compute. Allowed values are 0 (default, no property evaluation on a grid), 1 (electrostatic potential on a 2D grid), 2 (electron/spin density on a 2D grid), 3 (gradient of electron/spin density on

a 2D grid), 4 (Laplacian of electron/spin density on a 2D grid), 5 (molecular orbital values on a 3D grid). Since `spin_prop` is set, the spin density will be evaluated on a grid.

Grid specification is a little bit tricky but very flexible. `grid_origin` specifies the origin of the rectangular coordinate system associated with the grid in the refernence frame. `grid_unit_x` specifies a reference frame vector which designates the direction of the x-axis of the grid coordinate system. `grid_unit_y` is analogously a reference frame vector which, along with the `grid_unit_x`, completely specifies the grid coordinate system. `grid_unit_x` and `grid_unit_y` do not have to be normalized, neither they need to be orthogonal to either other - orthogonalization is done automatically to ensure that unit vectors of the grid coordinate system are normalized in the reference frame too. `grid_xy0` is a vector in the grid coordinate system that specifies a vertex of the grid rectangle with the most negative coordinates. Similarly, `grid_xy1` specifies a vertex of the the grid rectangle diagonally opposite to `grid_xy0`. Finally, `nix` and `niy` specify the number of intervals into which the x and y sides of the grid rectangle are subdivided. To summarize, the above input specifies a rectangular (in fact, square) 21 by 21 grid of dimensions 10.0 by 10.0 lying in the yz plane and centered at origin.

Running `oeprop` on such input will create a file called `sdens.dat` (for file names refer to man page on `oeprop`), which can be fed directly to `PlotMTV` to plot the data.

Specification of a three-dimensional grid for plotting MO isosurfaces (`grid = 5`) is just slightly more complicated. The index of the MO which needs to be plotted is specified by keyword `mo_to_plot`. The index is specified in Pitzer order and not according to the orbital energies. The reference frame is specified by keywords `grid_origin`, `grid_unit_x` and `grid_origin_y` (the third axis of the grid coordinate system is specified by by the vector product of `grid_unit_x` and `grid_unit_y`). Since in this case we are dealing with the three-dimensional grid coordinate system, one needs to specify two diagonally opposite vertices of the grid box via `grid_xyz0` and `grid_xyz1`. The number of intervals along z is specified via `niz`. The final input may look like this:

```
oeprop:(
  grid = 5
  mo_to_plot = 10
  grid_origin = (-5.0 -5.0 -5.0)
  grid_unit_x = (1.0 0.0 0.0)
  grid_unit_y = (0.0 1.0 0.0)
  grid_xyz0 = (0.0 0.0 0.0)
  grid_xyz1 = (10.0 10.0 10.0)
  nix = 20
  niy = 20
  niz = 20
)
```

Running `oeprop` on input like this will produce two files

- `mo.dat` - MO values tabulated in a column in order where x runs fast;

- `mo.pov` - the `MegaPov` command file which sets a number of parameters. You might have to adjust a number of parameters manually to obtain the best looking output.

Rendering of an image can be done via `megapovplus +Imo.pov`. Consult `MegaPov` documentation for more information on how to use it.

4.10 Utilities

4.10.1 `geom`

The program `geom` reads a set of Cartesian coordinates and determines from them the bond distances (Bohr and angstrom), bond angles, torsional angles, out-of-plane angles (optional), moments of inertia, and rotational constants. It requires either a `file11.dat` or `geom.dat` and writes `geom.out`.

5 `PSI3` modules

5.1 Overview of modules of `PSI3`

Below is a very brief list of some of the modules available in the `PSI3` program. For more information, consult the individual manual pages.

5.1.1 input

This program prepares the checkpoint file (`file30`). Errors will result if the `psi` program is run before `input`. This program reads the `default`, `input`, and (if given) `basis` sections of the input file and places molecule, geometry, and basis information into the checkpoint file. The molecule may be reoriented to a standard reference frame, and the molecular point-group symmetry is identified. It is possible to tell `input` to re-use orbitals from a previous calculation (`input --chkptmos`), although presently this only works if the point group has not changed.

5.1.2 cints

This program computes one- and two-electron integrals and derivative integrals. It is multi-threaded and has some advanced capabilities related to integrals-direct computations.

5.1.3 cscf

This program carries out the Hartree-Fock procedure. It can handle RHF, ROHF, UHF, and TCSCF. It can also simply re-orthogonalize previously existing MO's.

5.1.4 transqt

This module transforms one- and two-electron integrals from the symmetry-adapted atomic orbital (SO) basis to the molecular orbital (MO) basis for use in correlated computations by program such as `detci` or `ccenergy`. It can also back-transform one- and two-particle density matrices to the AO basis for contraction with the derivative integrals to obtain energy gradients.

5.1.5 ccsort

This module sorts the one- and two-electron integrals for use by the `PSI3` coupled-cluster programs.

5.1.6 ccenergy

This module computes the CCSD energy.

5.1.7 cctriples

This module computes the (T) correction to CCSD to give the CCSD(T) energy.

5.1.8 detci

This module performs many different types of CI computation, including CI's truncated according to substitution level (e.g., CISD, CISDT, CISDTQ, etc), full CI, and any CI which can be formulated as a restricted active space (RAS) CI, including second-order configuration interaction (SOCI) and other types of multi-reference CI's. It can compute one- and two-particle density matrices and can obtain CI natural orbitals.

5.1.9 detcas

This program works together with the `detci` program to do CASSCF calculations in a two-step procedure. It obtains the orbital gradient and rotates the orbitals to optimize them.

5.1.10 detcasman

This is a driver program that controls `detci` and `detcas` to perform CASSCF computations.

5.1.11 clag

This forms the CI lagrangian, which is needed for CASSCF and CI gradients.

A PSI3 Reference

T. Daniel Crawford, C. David Sherrill, Edward F. Valeev, Justin T. Fermann, Matthew L. Leininger, Rollin A. King, Shawn T. Brown, Curtis L. Janssen, Edward T. Seidl, Yukio Yamaguchi, Wesley D. Allen, C. Brian Kellogg, and Henry F. Schaefer III, PSI 3.2, 2002.

B Sample input.dat file

```
default: (
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Test input.dat for errors
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% check = true
  check = false

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% System parameters and job title
%%%%%%%%%%%%%%%%%%%%%%%%%%%
  memory = (50.0 MB)
  label = "CH3 cc-pVTZ SCF"

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Choose wavefunction
%%%%%%%%%%%%%%%%%%%%%%%%%%%
  wfn = scf
% wfn = mp2
% wfn = detci
% wfn = ccsd

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Derivative level
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% dertype = none
  dertype = first


%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Electronic structure info
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% reference = rhf
  reference = rohf
% reference = uhf
% reference = twocon
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Optimize geometry
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% opt = true
  opt = true
  nopt = 10

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Finite difference
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% disp = true
% ndisp = 1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Scratch files
% note: you must change psiuser
% to your username
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  files: (
         default: (
                  name = "ch3"
                  volume1 = "/tmp1/psiuser/"
                  volume2 = "/tmp2/psiuser/"
                  volume3 = "/tmp3/psiuser/"
                  volume4 = "/tmp4/psiuser/"
                  )
         file30: ( nvolume = 1 volume1 = "./" )
         )
)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Section that generate file30
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
input: (
  units = angstrom
  basis = ccpvtz
  zmat = (
   (x)
   (c 1 1.0)
   (h 2 0.97 1 90.0)
   (h 2 0.97 1 90.0 3  120.0)
   (h 2 0.97 1 90.0 3 -120.0)
  )
)
```

C Sample `basis.dat` file

```
basis: (
  NITROGEN:t2pd = (
    (S (  13520.          0.000760)
       (   1999.          0.006076)
       (    440.0         0.032847)
       (    120.9         0.132396)
       (     38.47        0.393261)
       (     13.46        0.546339))
    (S (     13.46        0.252036)
       (      4.993       0.779385))
    (S (      1.569       1.000000))
    (S (      0.5800      1.000000))
    (S (      0.1923      1.000000))
    (S (      0.06742     1.000000))
    (P (     35.91        0.016916)
       (      8.480       0.102200)
       (      2.706       0.338134)
       (      0.9921      0.669281))
    (P (      0.3727      1.000000))
    (P (      0.1346      1.000000))
    (P (      0.04959     1.000000))
    (D (      1.60        1.000000))
    (D (      0.40        1.000000))
                  )

 HYDROGEN:t2pd = (
    (S (       33.64            0.025374)
       (        5.058           0.189684)
       (        1.147           0.852933))
    (S (        0.3211          1.0))
    (S (        0.1013          1.0))
    (S (        0.03016         1.0))
    (P (        1.50            1.0))
    (P (        0.375           1.0))
                  )
     )
```